β−Prolog における Garbage Collection*

3J－7

田畑 宏治　周 能法　廣田 豊彦　橋本 正明†
九州工業大学情報工学部‡

1　はじめに

Prolog の処理系の基盤として標準的となってきた抽象マシン WAM (Warren's Abstract Machine)[1] は，極力" ごみ" データが少なくなるように工夫されているが，大規模な応用プログラムを扱うためには，やはり Garbage Collection (ごみ集め，以下 GC) が必要となってくる

Older らは，WAM におけるメモリ使用の特徴に着目し，ヒープの特別な領域に対して，頻繁に GC を行う方法を提案している [2].

著者らは，Prolog プログラムを，内部構造である照合木に変換し，ユニフィケーションの高速化を計っている β−Prolog の開発を進めている．これは，WAM を拡張した抽象マシン "NTOAM"[3] 上で実装されており，そのメモリ特性を生かした上記のアルゴリズムの導入と検証を行う．

2　NTOAM

NTOAM は，WAM をベースに実現されており，引数の受渡しなどにレジスタを使用しないなどの，高速化のための工夫を行っている．しかしメモリ使用の面では，WAM と大きな変化はなく，WAM の GC として考案された Older らのアルゴリズムが，そのままの形で実装可能である．以下に，GC に関わるデータエリア並びに外部参照についての紹介を行う．

2.1　データエリア

NTOAM には，データの論理的構造を保持するヒープと，バックトラック時に使用されるトレイルスタック，それらの制御を行うコントロールスタックがあり，それぞれヒープ，トレイルはセル，スタックはフレームの単位で構成されている (図 1).

データの構造は，セルのタグとポインタで実現され，変数のポインタは，それぞれのその値のはいっているセルを指す．自由変数は，自分自身を指すポインタで表現する．リストやストラクチャは，それぞれのタグが付けられたセルの指す先から，物理的に隣接したセルで，それらのメンバーを構成する．リストの場合，それは頭部と尾部の 2 つのセルで構成する．

トレイルは，バックトラック発生時に変数を元に戻すため，バインド時にその変数を蓄える．

スタックのフレームは，現述語の呼び出し側のフレームへのポインタ，述語の成功時と失敗時のプログラムの行き先，述語の引数，内部変数を格納し，それぞれの述語の呼び出しで生成され，帰還時，削除される．だだし，非決定的な述語呼び出しの場合，バックトラックを可能にするため，上の値の他に，現時点でのヒープやトレイルのトップ (それぞれ HB，TB とする) の値を格納し，チョイスポイントフレーム (以下 CP) と呼ぶ．また通常のフレームには，一番最近の CP (以下 LCP) へのポインタも格納する．

現時点での実行で失敗すると，バックトラックが生じ，スタックが LCP までリセットされ，そのフレームの内容からヒープもリセットされる．この時，トレイルの内容を参照することで，LCP が作られてからバックトラックまでにバインドされた変数を，元に戻すことが可能となる．

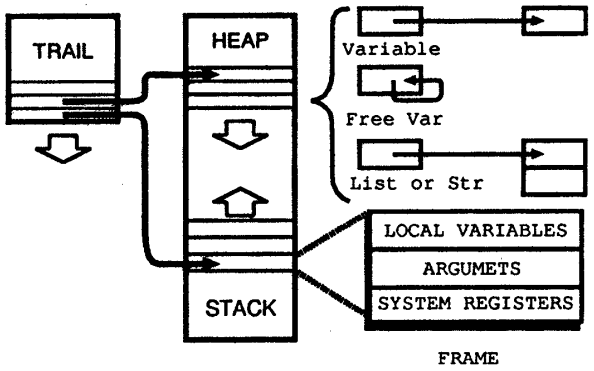

図 1: NTOAM のデータエリア

2.2　外部参照

巨大なメモリ空間のある小さな領域に対して GC を行うことの難しさは，その場所を参照可能なすべての外部参照を見つけ出さなくてはいけない事にある．しかし，NTOAM の CP やトレイルの技法 (WAM も同様) は，特別な処理を行わずに，これらすべての外部参照の探索を可能にしている．これは変数に，

*Garbage Collection for β−Prolog
†Koji TABATA　Neng-Fa Zhou　Toyohiko HIROTA　Masaaki HASHIMOTO
‡Faculty of Computer Science and System Engineering Kyusyu Institute of Technology

- スタックの変数はヒープのデータを参照可能だがその逆は許されない.
- ヒープの変数は原則として，新しいセルから古い方へ参照されるが，逆の場合トレイルに積む.

等の制約があるためで，GC の領域を LCP が生成されてから作られたヒープの領域，つまり HB からヒープのトップに制限すれば，その領域に参照するようなセルは，現在のフレームから LCP までの変数を走査すれば良く，その範囲外であっても，トレイルに積まれているはずである．つまり，LCP 以降のフレームとトレイルを走査すれば十分である．このことにより，走査データのもっとも少ない，現在の LCP からの帰還の時が，GC の機会に適しているといえる.

しかし新たにフレームが生成される場合，そのフレームを初期化することはなく，GC の時にフレーム中の各セルが有効であるかどうかの判断は困難である．そこで GC の実装に際して，新フレームは必ず初期化を行うように NTOAM の改造を行った.

3 アルゴリズム

ヒープのセルというのは，その論理的なまとまりを破壊しなければ，物理的な順序を保存する必要はなく，再配置可能である．そこで Older らは，具体的な WAM の GC の作成の際，外部から参照されたセルを，一時的な領域に移動させる方法を取り，それは以下に述べる基本的なアルゴリズムを用いている.

1. 外部から直接参照のあるセルを移動先の先頭へ移動し，最終的にそのセルが置かれる場所を指すよう，外部参照のポインタを更新する.
2. 移動先のセルを先頭から走査し，移動元を指すようなセルがあれば，次々その内容を移動し，同様にそのポインタを更新する．この走査を終了すれば，すべての必要なセルの移動が完了しているはずである.
3. 移動したセルはすべて HB から並べ，ヒープのトップを更新する.

ここでのセルの一時的な移動先には，GC の対象となる領域のサイズだけあれば十分なため，ヒープのトップからの空き領域を用いることができる．今回著者らは，このアルゴリズムを NTOAM へ実装した.

4 評価

4.1 評価方法

GC の回数を制限するため GC の対象領域のサイズを閾値とし，それより大きなサイズの場合のみ GC を行うようにする必要がある．そこで閾値を変化させることにより，その値が実行時間やヒープの最大サイズに与える影響を調べ，GC の性能を確かめた.

サンプルとして様々なサイズのパズルのプログラムを用いた．これらはバックトラックが多く，GC の性能が全体の処理時間に大きく影響する.

4.2 評価結果

8 個のサンプルを実行したが，このうち 5 個はほとんど GC を必要としなかった．GC を必要とした比較的大規模な 3 個のサンプルを調べた結果，5K ～ 10K の閾値を設定すると，実行時間のオーバーヘッドが 3% ～ 43% でヒープの最大サイズが 2% ～ 45% にまで減少することが確かめられた．サンプルのうち queens についての評価値を示す (図 2)．左右の縦軸は，GC を行わない場合との比率を表している.

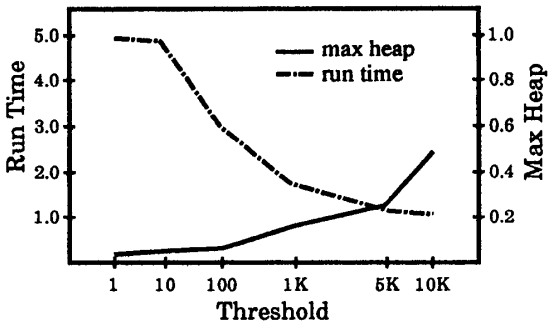

図 2: GC の評価結果 (サンプル queens の場合)

5 おわりに

GC には速さが要求され，それがシステムの性能を左右している．上記の結果をみても，閾値が 5K ～ 10K の場合でさえ実行時間に対する負荷が，無視できないものになっている．今後，GC の効率的な実装を検討する.

参考文献

[1] Hassan Ait-Kaci. *Warren's Abstruct Machine A Tutorial Reconstruction.* The MIT Press, 1991.

[2] William J. Older and John A. Rummell. An incremental garbage collector for wam-based prolog. In Krzyszof Apt, editor, *Logic Programing*, pp. 369–383, 1992.

[3] Neng-Fa Zhou. On the scheme of passing arguments in stack frames. In *Logic Programming*, pp. 159–174, 1994.